Sprawozdanie z projektu

Symulator tomografu komputerowego

Weronika Radzi 141303, Szymon Sroka 141312

1. Informacje ogólne

- Do składu grupy należy Weronika Radzi (numer indeksu 141303) oraz Szymon Sroka (141312).
- Zastosowaliśmy stożkowy model tomografu.
- Do zaprogramowania symulatora wykorzystaliśmy język Python wraz z Jupyter Notebook'iem.
- Dodatkowo użyliśmy dodatkowych bibliotek: numpy (do obsługi macierzy potrzebnych do obliczeń), skimage oraz matplotlib (do obsługi obrazów), pydicom (w celu dodania funkcjonalności obsługi plików DICOM), cv2 (do normalizacji wyników przy eksporcie do DICOM), datetime (do określenia daty i godziny symulowanego badania) oraz ipywidgets (do utworzenia interaktywnego interfejsu).

2. Opis głównych funkcji programu

a. pozyskiwanie odczytów dla poszczególnych detektorów

```
for angle in angles:
    #calculate start end points of rays
    start_x_coords,start_y_coords,end_x_coords,end_y_coords=[],[],[],[]
    for k in range(num_of_detectors):
        start_x_coord=radius*np.cos(np.deg2rad(angle))+img_centre_x
        start_y_coord=radius*np.sin(np.deg2rad(angle))+img_centre_y

        end_x_coords.append(radius*np.cos(np.deg2rad(angle+offsets[k]+180))+img_centre_x)
        end_y_coords.append(radius*np.sin(np.deg2rad(angle+offsets[k]+180))+img_centre_y)

    lines_coords=[]

    #apply Bresenham algorithm
    for line in range(num_of_detectors):
        lines_coords.append(BresenhamLine(int(start_x_coord),int(start_y_coord),
                                          int(end_x_coords[line]),int(end_y_coords[line])))


    #calculate brightness of each ray
    lines_brightnesses=[]

    for line in lines_coords:
        res=0
        for i in range(len(line[0])):
            if(line[0][i]<width and line[1][i]<height):
                res+=img[line[0][i],line[1][i]]
        lines_brightnesses.append(res/len(line[0])) #normalization

    #filter
    if(enable_filter): lines_brightnesses=np.convolve(lines_brightnesses, kernel, mode='same')

    sinogram.append(lines_brightnesses)
```

b. ustalanie jasności poszczególnych punktów obrazu wynikowego oraz jego przetwarzanie końcowe
(np. uśrednianie, normalizacja)

```
    #x -> lines_coords[i][0]
    #y -> lines_coords[i][1]
    #brigtness -> lines_brightnesses[i]
    output=np.zeros((width, height))



    for i in range(len(lines_coords)):
        for j in range(len(lines_coords[i][0])):
            x=lines_coords[i][0][j]
            y=lines_coords[i][1][j]

            if(x<width and y<height):
                output[x][y]+=lines_brightnesses[i]

    #normalization
    output/=len(lines_coords)

    #result of this iteration
    partial_output.append(output)

    print(".",end="")


print("\nstep 1/2 DONE")

#calculate result of all iterations
partial_avg_output=[np.zeros((width, height)) for i in range(len(angles))]

for i in range(width):
    for j in range(height):
        suma=0
        for iteration in range(0,len(angles)):
            suma+=partial_output[iteration][i][j]
            partial_avg_output[iteration][i][j]=suma/(iteration+1)

    if(i%10==0):print(".",end="")

#normalize final image and result of each iteration
for i in range(len(partial_avg_output)):
    cv2.normalize(partial_avg_output[i], partial_avg_output[i], alpha=0, beta=1, norm_type=cv2.NORM_MINMAX)
```

```
def radon_transform(filename, num_of_detectors,alpha_delta,offset,enable_filter):
    #load image
    img=io.imread(filename,as_gray=True)
    cv2.normalize(img, img, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX)
```

c. zapis plików DICOM

```
def write_as_dicom(pixel_array,name,patient_id,out_filename,comment):
    meta = Dataset()
    meta.MediaStorageSOPClassUID ='1.2.840.10008.5.1.4.1.1.2'
    meta.MediaStorageSOPInstanceUID = pydicom.uid.generate_uid()
    meta.TransferSyntaxUID = pydicom.uid.ExplicitVRLittleEndian
    ds = FileDataset(None, {}, preamble=b"\0" * 128)
    ds.file_meta = meta

    ds.is_little_endian = True
    ds.is_implicit_VR = False

    ds.SOPInstanceUID = meta.MediaStorageSOPInstanceUID

    ds.PatientName = name
    ds.PatientID = patient_id
    ds.ImageComments = comment


    ds.Modality = "CT"
    ds.SeriesInstanceUID = pydicom.uid.generate_uid()
    ds.StudyInstanceUID = pydicom.uid.generate_uid()
    ds.FrameOfReferenceUID = pydicom.uid.generate_uid()

    ds.BitsStored =16
    ds.BitsAllocated = 16
    ds.SamplesPerPixel = 1
    ds.HighBit = 15

    ds.ImagesInAcquisition = 1
    ds.InstanceNumber = 1

    ds.Rows= pixel_array.shape[0]
    ds.Columns = pixel_array.shape[1]

    ds.ImageType = r"ORIGINAL\PRIMARY\AXIAL"

    ds.PhotometricInterpretation = "MONOCHROME2"
    ds.PixelRepresentation = 0

    dt = datetime.datetime.now()
    ds.ContentDate = dt.strftime('%Y%m%d')
    timeStr = dt.strftime('%H%M%S.%f')
    ds.ContentTime = timeStr

    pydicom.dataset.validate_file_meta(ds.file_meta, enforce_standard=True)

    cv2.normalize(pixel_array, pixel_array, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX)
    pixel_array = pixel_array.astype(np.int16)

    ds.PixelData = pixel_array.tobytes()

    ds.save_as(out_filename, write_like_original=False)

    print("\033[1m saved! \033[1m")
```

d. odczyt plików DICOM

```
def open_file(filename):
    my_file=pydicom.dcmread(filename)
    print(my_file)

    #check bitsstored value
    converted=''
    if(my_file.BitsStored==16): converted=np.frombuffer(my_file.PixelData,dtype=np.uint16)
    elif(my_file.BitsStored==8):converted=np.frombuffer(my_file.PixelData,dtype=np.uint8)
    else: return

    img=np.zeros((int(my_file.Rows),int(my_file.Columns)))

    idx=0
    for i in range(int(my_file.Rows)):
        for j in range(int(my_file.Columns)):
            img[i][j]=converted[idx]
            idx+=1

    plt.imshow(img,'gray')


print("\033[1m Choose DICOM file to open: \033[1m")
interact_manual(open_file,
                filename="output.dcm")
```

e. filtrowanie sinogramu, zastosowany rozmiar maski

i. Zastosowaliśmy maskę składającą się z 21 elementów:

```
kernel_size=21
kernel=np.zeros(kernel_size)
center = kernel_size // 2
for k in range(kernel_size):
    kernel[k] = ( 0 if (k-center)%2==0 else ((-4/np.pi**2)/((k-center)**2)))
kernel[center] = 1
```

ii. filtrowanie:

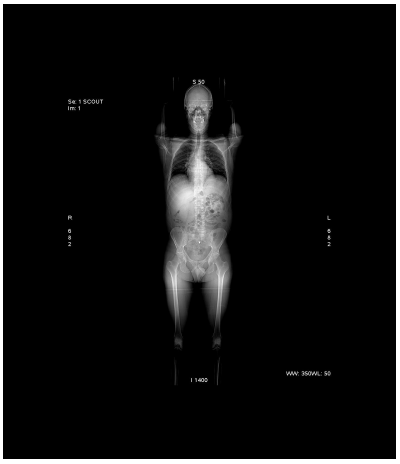
```
#filter
if(enable_filter): lines_brightnesses=np.convolve(lines_brightnesses, kernel, mode='same')

sinogram.append(lines_brightnesses)
```

f. wyznaczanie wartości miary RMSE na podstawie obrazu źródłowego oraz wynikowego

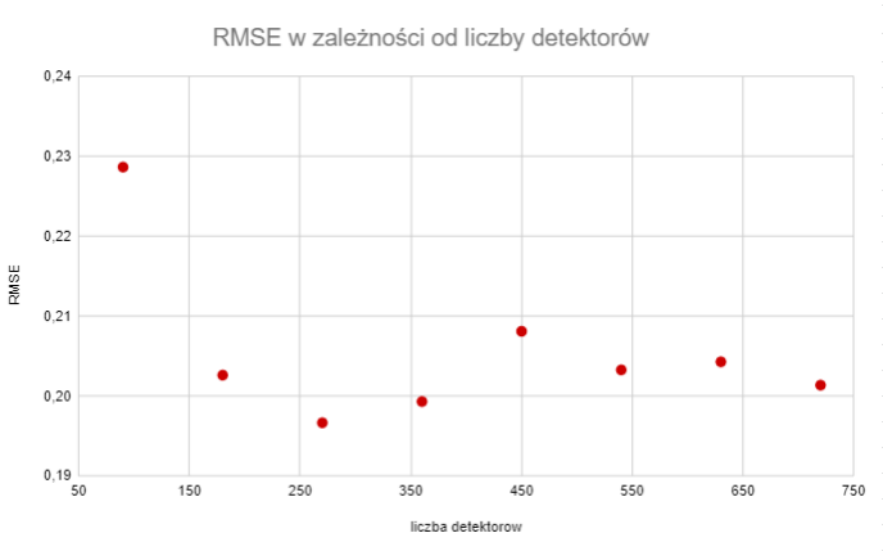
```
def show_result(img,sinogram,partial_avg_output,x):

    #take original image, normalized final output and calc RMSE
    print('RMSE=',np.sqrt(((img-partial_avg_output[x-1])**2).mean()))
```

3. Analiza statystyczna otrzymanych wyników

Nasze analizy przeprowadziliśmy na następującym obrazie:



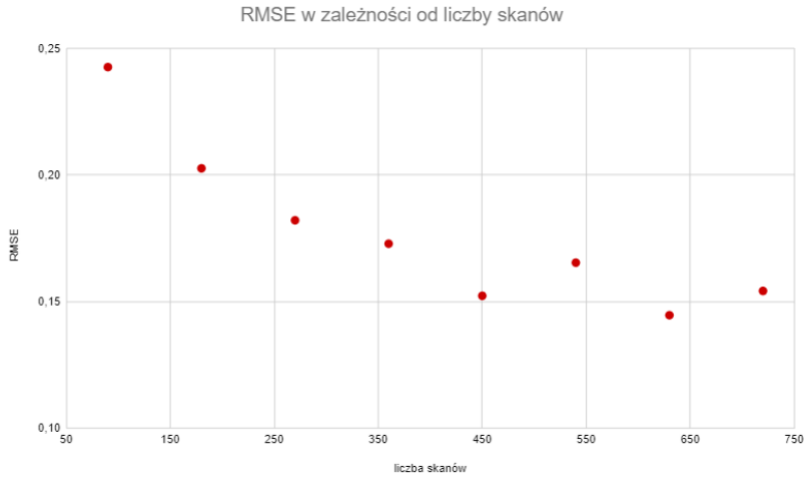
a. liczba detektorów zmienia się od 90 do 720 z krokiem 90



Wynik tego eksperymentu jest zgodny z naszymi oczekiwaniami oraz z subiektywną oceną jakości obrazu - im większa liczba detektorów, tym mniejszy (z drobnymi fluktuacjami) jest błąd odwzorowania obrazu oryginalnego. Wynika to z faktu, że przy większej liczbie detektorów ich zagęszczenie w obrębie ustalonej rozpiętości wachlarza jest większe, stąd więcej wiązek promieniowania przechodzi przez badany obiekt, co pozwala uzyskać większą dokładność.

Zauważamy szybki spadek błędu odwzorowania w zakresie od 50 do ok. 250 detektorów, przy następują fluktuacje, jednak subiektywna jakość obrazu jest bardzo dobra. Pozwala to przypuszczać, że do uzyskania zadowalających efektów pracy tomografu około 350 detektorów powinno być wystarczające.

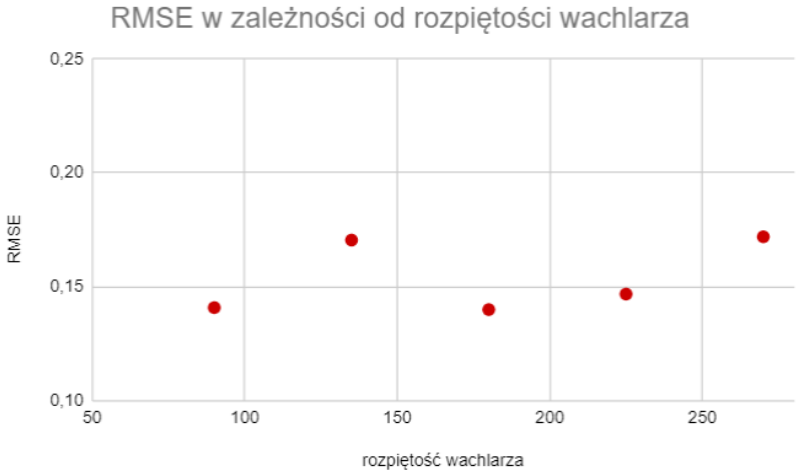
b. liczba skanów zmienia się od 90 do 720 z krokiem 90



Zauważamy, że liczba skanów, podobnie jak liczba detektorów, ma duży wpływ na jakość uzyskanego obrazu: ich większa liczba sprawia, że błąd RMSE staje się (poza drobnymi fluktuacjami) coraz mniejszy.

Większa liczba skanów oznacza większą precyzję w przechodzeniu po okręgu, po którym poruszają się detektory, a to wpływa na lepszą wierność odwzorowania obrazu oryginalnego. Uzyskane wyniki są zgodne z subiektywną oceną uzyskanych obrazów wyjściowych; coraz większa liczba skanów pozwoliła na wykrycie większej ilości szczegółów badanego obiektu.

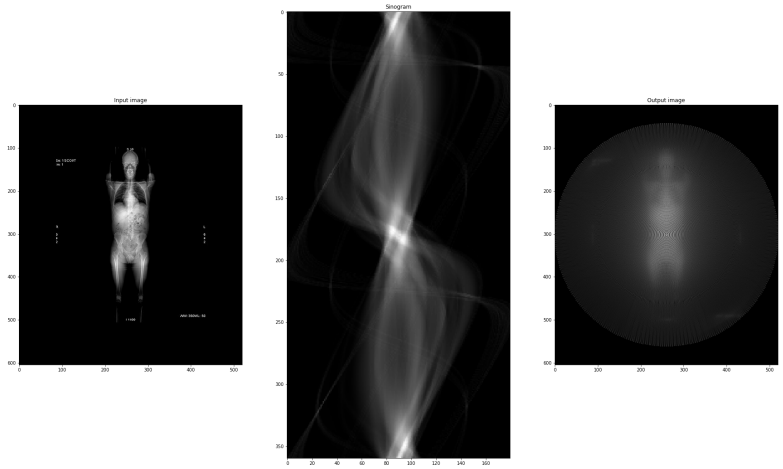
c. rozpiętość wachlarza zmienia się od 45 do 270 stopni z krokiem 45 stopni



W przypadku zmian rozpiętości wachlarza zauważamy wzrost błędu przy wzroście rozpiętości detektorów z drobnymi fluktuacjami. Z początku taki wynik był dla nas zaskoczeniem, jednak po przeanalizowaniu problemu stwierdziliśmy, że taki wzrost błędu może być skutkiem rozmieszczania stałej liczby detektorów na coraz większej rozpiętości, a to przekłada się na mniejszą dokładność odwzorowania obrazu oryginalnego. Uzyskany wynik jest zatem zgodny z teoretycznym rozważaniem, jednak - również przy badaniu wpływu tego parametru na jakość innych obrazów - nie zawsze szedł on w parze z subiektywną oceną jakości obrazu.

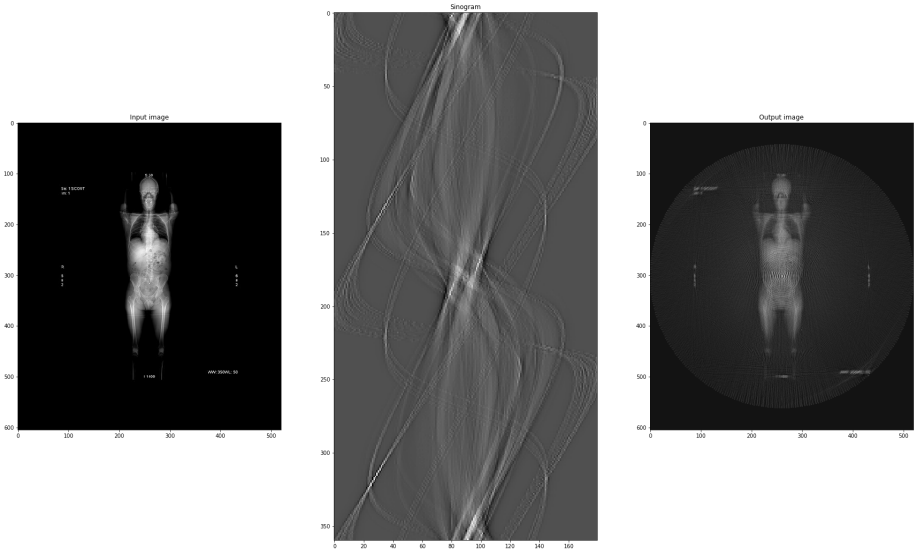
d. porównanie działania programu z włączonym i wyłączonym filtrowaniem

Bez filtrowania:



RMSE=0.12134067809564046

Z filtrowaniem:



RMSE= 0.13340768993755164

Mimo, że po włączeniu filtrowania wartość RMSE wzrosła, algorytm z uwzględnionym filtrowaniem znacznie lepiej odwzorowuje oryginalny obraz. Na przefiltrowanym obrazie widoczne są szczegóły, których nie sposób dostrzec na obrazie bez filtracji - bardzo dobrze widoczne są elementy szkieletu i organów, a nawet widoczny jest tekst, który był umieszczony przy konturze obrazu oryginalnego. Kernel o rozmiarze 21 okazał się wystarczający do osiągnięcia zadowalających efektów działania tomografu.